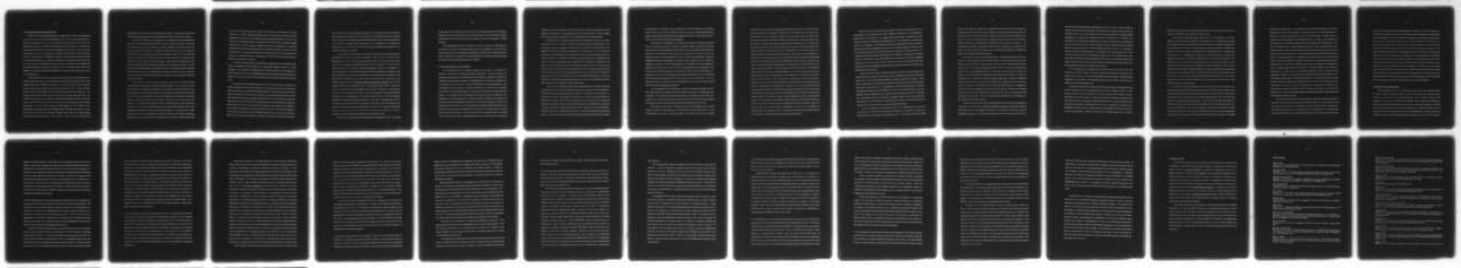
AD-A076 417 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 5/2
ON DATABASE MANAGEMENT SYSTEM ARCHITECTURE.(U)
OCT 79 M HAMMER, D MCLEOD N00014-76-C-0944
UNCLASSIFIED MIT/LCS/TM-141 N/L

1 OF 1
AD A 076417



END
DATE FILMED
1-81
DTIC

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-141

ON DATABASE MANAGEMENT SYSTEM ARCHITECTURE

Michael Hammer

Dennis McLeod

October 1979

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract number N00014-76-C-0944 and by the Joint Services Electronics Program (University of Southern California).

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TM-141	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: On Database Management System Architecture	5. TYPE OF REPORT & PERIOD COVERED: Interim rept.
	6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TM-141
7. AUTHOR(s): Michael Hammer and Dennis McLeod	8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0944
9. PERFORMING ORGANIZATION NAME AND ADDRESS: MIT/Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 12 43
11. CONTROLLING OFFICE NAME AND ADDRESS: ARPA/Department of Defense, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: October 1979
	13. NUMBER OF PAGES: 41
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: ONR/Department of the Navy, Information Systems Program, Arlington, VA 22217	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document has been approved for public release and sale; its distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

database
management
database management system architecture
conceptual data models
database semantics

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Despite the many advances that have been made in the field of database management in the last two decades, in many respects the paradigm of database management has not changed much since its inception. Several long-standing assumptions pervade the field and exert a great influence on the architecture of database management systems, their functions, and the kinds of databases that they manage. This paper reconsiders some of these assumptions and suggests certain alternatives to them. In particular, it is argued that the concept of an integrated database ought to be supplanted by that of a federated database

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. a loose assembly of semi-independent components; the position of the database management system in the context of a total information system is reexamined, and arguments are made for extending its functional capabilities; and controlled logical redundancy in the schema is introduced as a means of improving the usability of a database and of enhancing its life-cycle performance. An underlying theme throughout is that of the importance of a semantic schema of the database, which specifies enough of the meaning of the application domain to enable enhanced functionality to be achieved. A number of characteristics of a conceptual data model (in which this schema would be expressed) are described.

ON DATABASE MANAGEMENT SYSTEM ARCHITECTURE

Michael Hammer
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Dennis McLeod
Computer Science Department
University of Southern California
Los Angeles, CA 90007

ABSTRACT:

Despite the many advances that have been made in the field of database management in the last two decades, in many respects the paradigm of database management has not changed much since its inception. Several long-standing assumptions pervade the field and exert a great influence on the architecture of database management systems, their functions, and the kinds of databases that they manage. This paper reconsiders some of these assumptions and suggests certain alternatives to them. In particular, it is argued that the concept of an integrated database ought to be supplanted by that of a federated database, a loose assembly of semi-independent components; the position of the database management system in the context of a total information system is reexamined, and arguments are made for extending its functional capabilities; and controlled logical redundancy in the schema is introduced as a means of improving the usability of a database and of enhancing its life-cycle performance. An underlying theme throughout is that of the importance of a semantic schema of the database, which specifies enough of the meaning of the application domain to enable enhanced functionality to be achieved. A number of characteristics of a conceptual data model (in which this schema would be expressed) are described.

KEYWORDS AND PHRASES: database management, database management system architecture, conceptual data models, database semantics.

This research was supported by the Advanced Research Projects Agency of the Department of Defense - monitored by the Office of Naval Research under contract number N00014-76-C-0944, and by the Joint Services Electronics Program (University of Southern California).

1. INTRODUCTION

Database management systems represent a successful and dynamic technology. From a modest beginning, they have grown to be the most important software development of the 1970's. Today's database systems provide broad ranges of function and exhibit high degrees of performance that are far removed from the primitive file systems from which they have evolved. Database management research is a vital and growing area of computer science [Hammer 1979], and contemporary research prototypes promise even greater capabilities in future commercial products. Nonetheless, despite all this activity and despite the many advances that have been made in the last two decades, we remain in the first generation of this branch of software technology. In many respects, modern database management systems are based on a twenty-year-old paradigm of the appropriate nature and function of such systems, a paradigm that was formulated at a time when the context of computing and computer-based application systems was very different from what prevails today. As a result, the architectures of contemporary database management systems are heavily influenced by tacit assumptions and perspectives that are often irrelevant or obsolete. Even important innovations in the field [Codd 1970] and proposed future architectures [Nijssen 1976] are formulated within a conventional framework for the function and structure of database systems.

In this paper, we reexamine a number of the fundamental assumptions regarding the nature of the database management function and consider some alternatives to the standard perspectives on several important issues. We shall consider the implications of these views for DBMS architecture, and derive a number of criteria that should be satisfied by a future generation of these systems.

2. DATABASE ARCHITECTURE

Our starting point will be the concept of a database itself, for its definition ultimately determines the nature and architecture of database management systems (DBMS). To search for a precise formulation of this concept with which all would agree is an endless and essentially futile task. However, it is possible to identify the major ideas that the term currently connotes. Most generally, of course, a "database" simply means a computer-based collection of information. However, the term is usually interpreted much more narrowly; a number of additional restrictions are imposed on the databases that contemporary database management systems are prepared to handle. First, databases are generally limited to "structured" or "formatted" data; i.e., they consist of discrete records, each containing a finite number of atomic fields. Examples of collections of information that do not satisfy these requirements are text files and aggregates of signal data; both of these are examples of "continuous" or "unstructured" data. Second, it is assumed that a contemporary database is much larger than its description, that it contains a comparatively small number of different kinds of data, but many instances of each kind. (In other words, the number of record types is much smaller than the number of record occurrences.) This is in contradistinction to the so-called "knowledge bases", used in artificial intelligence applications, which are also discrete and formatted, but which have a different ratio between description and content. In these knowledge bases, there are typically many different kinds of data, with only a very few (often one) instances of each kind; in practice, to prevent the description from swamping out the data, the description of a data item becomes part of the item itself, so that every entry in the knowledge base is viewed as an instance of a lowest common denominator (such as a "fact").

Fundamentally, a database is meant to capture and express information about an application; it should serve as a model of that application. Yet, in the database community,

sharp distinctions are drawn between three different kinds of information: data, which is manifested as the explicit contents of a database; explicit knowledge about an application, which is expressed in the schema for the database; and, implicit knowledge about the application, which may not be captured anywhere other than in the minds of the designers and users of the database. For example, the fact that a given employee is 30 years old is expressed in the database; the fact that every employee must have an age is expressed in the schema; and the fact that employee age is usually between 25 and 60 may not be expressed anywhere, and is viewed as part of the "application semantics" presumably known to all who use the database.

These (artificial) distinctions are motivated by the same factors that led to the previous assumptions regarding database structure; they derive from the nature of the data processing systems prevalent in the 1960's. Uniformity in data and its handling was necessary at that time in order to achieve the economy of scale and the efficiency of processing demanded by large, single-processor systems. However, as we shall argue below, the context for database applications is changing in such a way as to call for the revision of this perspective.

Over and above these implicit assumptions about contemporary databases is a very explicit one that is based on what is probably the major concept in the "database" approach to information systems: integration. This principle states that all data used by a family of related, though distinct, applications should be unified into a single collection of data. (This contrasts with the practice of many conventional information systems.) Another way of expressing this idea is that data ought to be viewed as an autonomous organizational resource, rather than as the property of some particular program or user. This "logical centralization" of data decreases the degree of data redundancy in the total system, thereby diminishing opportunities for data inconsistencies and related errors; it enables the more

ready implementation of applications that require data from several sources; and, it establishes uniform modes of access and usage for all the data. The benefits that accrue from this approach are very real, and are responsible for a phenomenal rise in the use of database systems in the last decade. However, it must be realized that data integration also imposes certain limitations on information systems; these limitations have associated costs, and they ought to be articulated and examined.

The principal difficulty with the "database" concept is that, in reacting to the fully decentralized situation that prevailed in the pre-database world (with all its attendant problems), "integration" may go too far in tightly coupling together aggregates of data that ought to retain some individual autonomy. Many of the problems that this can raise are illustrated by the consequences of appointing a database administrator (DBA) for an integrated database. The role of the database administrator is inherent in the integrated approach to database management. The DBA is a central authority responsible for constructing and managing "the" database. He maintains control over all the data, and is responsible for determining and adjudicating the disparate needs of the users in his community. Therefore, the ultimate providers and users of the data must cede their authority over it to the DBA, who is outside their own organization. In practice, people (reluctantly) accede to this demand, either because of external pressures or because the benefits that they perceive in so doing outweigh the disadvantages. But these disadvantages are real, and alternatives, if feasible, might be desirable.

The disadvantages of a logically centralized database relate to issues of security and privacy, of performance, and of responsiveness to changing user needs. In the first case, users are often hesitant to entrust their private data to an external authority, despite any assurances they receive regarding its safety; their fears may or may not be valid (and experience shows that often they are), but they are real nonetheless. Generally, a local

producer or consumer of data, in ceding his control over it to a central authority, sacrifices his own needs for the overall benefit of the community; while perhaps noble, such sacrifice ought not be demanded if it is not necessary. Similarly, in the process of selecting a physical design for the database, the DBA must ascertain the global uses and response requirements of the database and then select a design that provides the best overall performance; while this design may achieve a global minimum, it may also be far from a locally optimal point for any individual user, who would prefer to see that part of the database with which he is concerned designed in a way to meet his private needs. That is, a compromise design that endeavors to satisfy all users may end by satisfying none of them. Similar remarks apply to the logical schema design process, and the extent to which the design eventually selected will naturally support a user's needs in programming his applications. Finally, by removing local control over the database, changes in the context and semantics of the data that originate with a user must be channeled through a remote DBA before they are reflected in revisions to the database structure. This indirection can introduce serious delays and inconsistencies.

We believe that a middle road should be followed between the rigid centralization of conventional database systems and the anarchy of completely diffused and decentralized files. We have termed this alternative a "federated database". (This term first originated, to the best of our knowledge, in a discussion between one of the authors and Prof. Maurice Wilkes of Cambridge University.) A federated database consists of a number of logical components, each having its own (conceptual) schema. These components are related, but independent; they may or may not be disjoint. Typically, a component of a federation will represent that collection of information needed by a particular application or closely related set of applications. Some of this information may also be needed by another application; in that case, it will also be described in the schema for another component. The individual

components are tied together by a higher-level schema that enables users of one component to access another without having a detailed knowledge of the other's schema. A component is a logical, not a physical, entity: a federation seeks to achieve partial logical decentralization; this may or may not be combined with physical distribution.

A user will most commonly be affiliated with one component of a federation, and his database transactions will generally retrieve and modify this "local" data. On occasion, however, he may need to access data that belongs to another component. Consequently, each component will present two interfaces to its users: one to its most frequent (and consequently most knowledgeable) users, and one to those whose primary affiliation lies elsewhere and who utilize this component only occasionally. The "local user" interface will correspond to a detailed and robust conceptual schema. The schema and physical storage structure for a component will be selected, and will evolve, so as best to satisfy its most frequent users. The infrequent user will be able to access a component without having to be familiar with the details of its logical structure; he will interact with it at a higher level of abstraction than he does when using his "own" data. The penalty that will be paid by the non-local user will be a reduction in his capability and flexibility, as well as in the efficiency with which the system processes his requests. The non-local interface will also shield occasional users from knowledge of changes to the component's local schemas. Each component guarantees to maintain its non-local interface, and so change and evolution are permitted so long as this semi-opaque interface continues to be supported. Furthermore, the non-local interface will allow only limited access to sensitive parts of the local data. In other words, a "two-tier" system is established, with different classes of users; an individual will generally be a "first-class" user of a small number of components, and a "second-class" user of the rest. Power and efficiency are delivered to those who need it, while ease of access is provided to others; these conflicting goals can be met by the provision of two levels of

interface to each component of the federation.

The concept of a federation is based on the observation that in practice, despite the availability of an integrated database, most functional units of an organization make use of only a subset of the total schema and a limited portion of the data. In such cases, the remainder of the database can actually be a burden to a user; he pays various costs for centralization, while receiving comparatively little in return.

Each component of a federated database may be under separate control and authority; its owner/administrator has the responsibility for constructing the description that will be seen by the component's primary users, and selecting a storage structure that will best satisfy them. Presiding over all these local DBA's is a "master" DBA, who has certain responsibilities and authority for the total collection of data. His functions principally relate to the interfacing of the independent components; they supplement, rather than conflict with, the activities of the local DBA's. The principal responsibility of the master DBA is to establish the non-local interface that each component will have to maintain. Another of his duties will be to determine how redundancy in the federation ought to be handled. Several components may need to make use of the same data; in some cases, it would be appropriate for only a single copy of it to be stored, with all users accessing it; in others, it would be better for each component to maintain its own copy, with a variety of possible consistency restrictions established to ensure that the various versions remain appropriately related. (It need not be the case that all copies will have to be kept entirely consistent; while that is one possible alternative, it has associated costs; less expensive alternatives should be available as well.) It should also be observed that this modular approach to database architecture is consonant with modern approaches to the engineering of large software systems. "Information hiding" through the use of semi-opaque interfaces enables the construction of large and complex systems that are reliable and flexible [Parnas 1972].

The concept behind a federated database is that local autonomy and primacy can be maintained without preventing appropriate forms of general data availability. Primary control over a database component will be vested in its principal maintainers and users, but adequate centralized authority will be exercised in order to ensure appropriate levels of consistency and to enable uses to access multiple components of the database when necessary. We believe that this architecture naturally models typical patterns of use found in contemporary integrated databases. Our approach also has the advantage of largely eliminating the herculean task of devising a single, detailed schema for an enormous, complex database. Each component of the database, which will usually be of a manageable scale, will have its own set of schemas, as developed by its own DBA. The master DBA will need only a general familiarity with each of the components, enough as to be able to establish their interfaces and monitor their ongoing relationships.

The idea of a federation ought not to be confused with that of a "distributed database". This latter term has already acquired a variety of meanings, most of which are based on the notion of maintaining logical centralization while achieving physical decentralization. That is, the database is still seen by its users as an integrated whole; however, it may be physically implemented in such a way that parts of it are stored at different sites in a network and/or are physically replicated. The major motivations for such distribution are to achieve better performance by higher degrees of concurrent operation, by locating data storage near its sources and uses, and by providing a higher degree of reliability and survivability. The concept of a federated database is unrelated to geography. The components of a federated database might all be collocated at a single site, or they might indeed be distributed in a network; the critical issue is that they are logically viewed as loosely coupled, rather than as tightly integrated.

3. DATABASE SYSTEM ARCHITECTURE

Another issue that needs reexamination and reconsideration is that of the appropriate nature and function of a database management system. In its current configuration, a database management system is a limited and passive software system, with narrow and well-defined functions. A DBMS has responsibility for the storage of a database, for providing users with access to it, and for managing these users in such a way as to ensure the integrity of the data. The position of a DBMS in the context of a total information system is a subsidiary one. The DBMS is responsible for the most mechanical tasks related to storing and retrieving data; it performs its functions only when explicitly instructed to do so by other more active components of the overall system. The semantics of the application are usually embedded in the logic and code of the rest of the system, rather than in the DBMS, and the internal operation of the DBMS is rarely based on the meaning of the data or its intended uses.

Classical data management is only one of the activities that must be performed by a modern information system. In current practice, related functions are provided by system modules other than the DBMS and make use of a variety of facilities (such as general purpose programming languages, data communications systems, etc.). A major difficulty with this kind of system architecture is its lack of naturalness and semantic coherence. The various systems and tools that are used to produce the final information system have not usually been designed so as to interface in a convenient and integrated fashion. The system builder must familiarize himself with a disparate set of languages, programming conventions, etc., all in order to address a single application. (For example, many contemporary programmers must know COBOL, CICS, and DL/I.) Furthermore, a system built in this way usually lacks a dominant component, which would give a natural shape and organization to the system as a whole; rather, it is likely to consist of a number of

disconnected subsystems, which are roughly patched together. In consequence, such systems are often poorly engineered, and therefore unreliable and costly to maintain and change.

Many of these problems can be alleviated if it is recognized that the database lies at the core of an information system and that the system's architecture ought to reflect that fact. Rather than viewing the database as an ancillary repository of information to be used by other functional components, it is possible to interpret all functions of the information system in terms of data management; the dog at last begins to wag the tail. This approach leads to a consistent and uniform framework in terms of which to conceptualize the activities of the total system and to effect its design. In other words, we believe that in data-intensive applications, the database system ought to assume a position of primacy. The role and scope of the DBMS ought to be expanded so that, rather than being a minor subsystem of a total information system, the DBMS will be its major component. As such, it will be in control and will be responsible for performing most functions, calling specialized subsystems only when necessary.

The functions that can be brought under the control of the DBMS include transaction processing, report writing, and exception handling. The first of these entails such activities as: formatting and displaying a screen, leading a clerk through data entry, and applying simple edit checks along the way; managing the transactions that multiple users are concurrently running on the system; and applying more sophisticated consistency checks to data input, to ensure that it is plausible in the context of the existing data in the database. At its simplest, report writing is merely a mild generalization of query processing, in which a set of records is identified, particular fields selected (perhaps with simple computations performed on them), and the result displayed in a specified format. More generally, this term can be extended to include most forms of data-intensive application programming, in which the amount of computation per unit data is comparatively small,

and in which the principal flow of control in the program is determined by the natural structure of the data. Exception handling refers to the detection of unusual situations, anticipated or not, and the subsequent taking of appropriate responses. Much of the activity of an information system is concerned with such extraordinary situations, ranging from data error detection (semantic integrity maintenance) to identification of situations defined to be of special interest (alerting) to the recognition of patterns or trends in the data. In all these cases, the principal issue involved is examining some part of the database in the context of a (semantic) model of the application, which defines the standards applied to detect any deviation from the norm.

In current practice, none of these functions are implemented by the database system. They are handled by other subsystems or by "application" programs written in a "host" language. Yet in all cases, the performance of the function demands an understanding of the meaning of the data. If the DBMS possessed a schema that described the semantics of a database as well as its structure, then it could naturally be assigned responsibility for these tasks.

In current practice, the DBMS does make use of very limited semantic information to perform rudimentary versions of these functions. For example, certain kinds of invalid data and transactions can be automatically detected because they violate the primitive assertions about the semantics of the data that are contained in a conventional schema (such as field data type, identity of keys, set cardinalities, and the like). There is no qualitative distinction between these and the more subtle and sophisticated error detection procedures applied by hand-coded edit routines. In all cases, some knowledge of the meaning of the database is being exploited to rule on the validity of a database transaction. If the schema contained a richer body of information about the database's meaning, then all this activity could be uniformly and consistently conducted by the DBMS. Similarly, current information systems

detect interesting or unusual situations by making use of a model of the application domain that is "procedurally embedded" in a set of programs. If this model were explicitly recorded in the schema, then the DBMS could assume responsibility for the automatic application of this knowledge to accomplish the tasks in question. The advantages of this "declarative" approach are several and relate to the improved system modularity, modifiability, and understandability that it provides.

It is possible to view most report writing and database application programming as generalized extensions of database manipulation. When the data is central to the application, then the structure of the application system, and in particular the control structures of its programs, should follow the structure of the database. To this end, the linguistic facilities for expressing the application program ought not be embedded in a general purpose processor-oriented language, but should be provided as extensions to the data manipulation constructs used to retrieve from and make updates to the database. We believe that a DBMS ought to provide, not a few constructs to be called from a host language program, but an integrated database programming language, used for the complete construction of application systems. This language should contain the operators necessary for performing basic computations on data values; furthermore, it ought provide data-driven control structures that allow for the natural description of algorithms typically encountered in such applications. For example, high-level iterators and rich exception signalling and handling mechanisms are likely to be desirable. The operational view of data objects may also prove useful; as expressed in recent work in abstract data types, this approach is based on describing a data object not in terms of its internal structure and representation but in terms of its behavior [Hammer 1976, Liskov+Snyder+Atkinson+Schaffert 1977, Schmidt 1977, Stonebraker 1977].

The virtues of an integrated database programming language are several. Principal

among them are the resulting natural structure of the application programs, the semantic coherence that derives from the use of a single language, and even potential efficiency improvements that follow from not having to repeatedly cross the host language - DBMS interface.

The foregoing indicates the directions in which we believe the capabilities of contemporary database management systems ought to be extended. In each case, the extension can be seen as rationalizing and generalizing facilities that are already provided. What is required is a willingness to break historical barriers and reexamine long-standing assumptions regarding the appropriate role of a DBMS.

4. THE LIFE CYCLE OF A DATABASE

Every software system is a dynamic, almost a living, entity. It is conceived in optimism. When born, it rarely meets anyone's expectations. It grows to maturity in changing circumstances and must continue to adapt to them for its entire life. Eventually, it succumbs to obsolescence and old age. The implications of this life cycle are manifold, and its significance in the context of a database system deserves careful consideration. The conventional view of this process is, of course, the following. Once the need for a database is perceived, a database administrator is commissioned to undertake its development and design. He is responsible for identifying the data requirements of the application, for producing an overall description of a database (conceptual schema) that accurately and naturally models the application environment, for describing the application views of the database (external schemas), and for selecting physical realizations of these schemas that will provide the required level of performance in the context of the transactions that will be conducted with the database. After the database has been installed, the DBA is charged with its maintenance, which is generally perceived to have two components: evolving the

conceptual and external schemas to reflect changes in the structure and semantics of the application environment; and reorganizing the internal schema in order to match changes in the usage pattern, the particular mix of transactions conducted with the database.

We believe that it is important to emphasize two issues that are relevant to the idea of the life cycle of a database and that do not receive explicit consideration in the prevailing model just summarized. The first is that two distinct sets of criteria need to be applied in the process of database development and maintenance: the first of these is naturalness and completeness of representation, the extent to which the database is an effective model, and a snapshot of its contents an accurate image, of the application; the second criterion is performance, which is determined by the degree of support that the design of the database gives to its most common and important uses. The second issue revolves around the concept of performance itself. The costs of a database in terms of space (for storage) and time (for transaction processing) are not the only ones that define its performance, and may no longer even be its most important components; the costs of building and maintaining the application programs that make use of the database must also play a prominent role in selecting designs for the database.

The criteria of modelling and performance may potentially be in conflict, since they are instances of even more general concepts that often are at variance: statics and dynamics, flexibility and functionality, generality and specificity. In focusing on the database as a representative model of the application, rather than on its role in the implementation of transactions, the DBA avoids making a commitment to any one particular way of using the database; by seeking to identify the fundamental semantics of the application, he produces a design that ought to equally well support all users. However, anything (including a database design) that is equally good for all uses is also not particularly good for any of them. A lack of commitment to a particular mode of use achieves generality at the expense

of convenience. On the other hand, an approach to database design that is extremely sensitive to the way in which the database is used by a particular class of applications will produce schemas much more useful for the applications in question. Achieving this will, of course, result in reduced flexibility and adaptability.

The importance of adaptability in a schema follows from the inevitability of change. There are two principal sources of change in information systems: revisions to the nature and characteristics of the application world, which in turn require enhancements to existing programs or the creation of new software; and changes to the ways in which the information system is used, expressed in terms of performance requirements, transaction frequencies, and operating priorities. As observed above, change can have a variety of impacts on a system. It is widely recognized that it is necessary to tune and reorganize a database's physical structures in order to meet its changing performance requirements. But the costs of transaction processing and data storage are only two of the components of a database's life cycle costs; increasingly, they are being dominated by the costs of building and modifying the application software that makes use of the database. And to a significant extent, these software costs are determined by the structure of the (external) schema used by the application programmer.

A given application can be described by a number of different, equivalent schemas. Obviously, some of these schemas will be more natural static models of the application than will others. On the other hand, some schemas will be more conducive to the writing of a program with a specified function than will others; furthermore, those that express the best static model are not necessarily those that most effectively support its dynamic uses.

Any program that accesses a database makes use of information that is either explicitly present in the database or that can be derived from some that is. If the information that is needed is explicitly available in the database and so can be readily

obtained by the program, then the program's complexity is reduced; if, on the other hand, the information must be computed in complex ways from that which is described by the schema, then the user program is likely to be more difficult both to construct and understand. Consider, for example, a database concerning ships; in each ship record, there are fields for the ship's name, captain, position, fuel level, and destination. A program that needs the position of a given ship can simply identify the name of the desired ship to the database system and specify that it wants its associated position. On the other hand, if the program needs to know the names of the captains of all ships with the same destination as a given one, then the complexity of the retrieval operation increases dramatically (depending on the retrieval facilities provided by the DBMS being used). Namely, the specified ship must be identified, its position field extracted, a connection made to all other ships with that value in its position field, and the captain fields of those records extracted. (The connection might be made by means of a join, or by following a link in an owner-coupled set, or by an explicit sequential search, depending on the system being used; in all cases, the connection must be explicitly made.) This is clearly more complex than the first example cited. However, it must be realized that this complexity is not inherent in the data being retrieved, but in its relationship to the schema that is available for use. If, for example, each ship record possessed a (multi-valued) field that listed all the captains of all the ships having the same destination as the ship in question, then the foregoing retrieval would actually be quite simple. Clearly, the difficulty of writing a database application program is dependent on the extent to which the data that it needs is easily accessible, and on the logical complexity of the code needed to retrieve this data. Therefore, the design of the database schema can have a major impact on the difficulty of construction, and hence on the reliability and maintainability, of application software. The difficulty of the database programmer's job is, to a substantial degree, determined by the database schema designer.

Thus, the process of database design ought to entail the identification of the transactions that are expected to be most frequently performed and the subsequent specification of the schema so as to support their programming. The designer of the schema can localize and highlight that information that is most frequently used and thereby enhance its availability. If, on the other hand, he does not perceive the ways in which the database will be used, programmers may have to write complex sequences of code in order to extract the information they need from that which is provided. In other words, there is a tradeoff between the richness of the schema and the complexity of the programs that make use of it; if information is explicitly available in the schema, then it need not be respecified in detail each time it is used. There is a strong interaction and duality between data definition and data manipulation, between what is expressed in the schema and what must be performed by transactions.

Thus, the dynamic view of a database, expressed in terms of its most frequent uses, should influence the design of the schema; it may exert different pressures from those that derive from a static model-oriented approach to database semantics and design. While these two approaches are not inherently in conflict, we believe that some distinction between them ought to be maintained. It is inappropriate to claim that the semantics of a database is entirely determined by its most frequent uses. There is an invariant semantic core to any database, which is embodied in the schema and which determines the set of transactions that can possibly be made with the database; adapting this core to the needs of a particular usage pattern is one of the major challenges facing the DBA. The importance of the semantic core is that it provides an approach to coping with change.

As a new application of the database arises, it may encounter the fact that the information that it requires, while available in the database, is not expressed in a form that facilitates the ready construction of the needed software. If the original schema is totally

tuned to the needs of an earlier set of requirements, then there may be no facilities for adapting it to these new needs; tuning a database so as to precisely meet existing needs can interfere with the handling of new ones. Of course, the old schema could be replaced by a new one, but that would disturb earlier applications that depend on the old schema. Change can not be accommodated by invalidating existing systems and software. An organization's inventory of application systems represents a significant investment, which cannot be easily written off. Change must be accommodated gracefully, by building on rather than invalidating what lies in the past.

We believe that, in order to achieve these goals, a schema must reflect both the static semantics of the application and its dynamic needs; we further believe that the most effective way of allowing a database design to evolve in order to meet changing needs is *through growth, by directly incorporating into the schema* the version of the database contents that a new application requires. In other words, we envision that a schema will contain a "base" that reflects the (relatively) fixed semantics of the domain, as well as a component that is tailored to the needs of particular applications. This latter component does not remain fixed; as new uses of the database evolve, additions will be made to this part of the schema. These additions are not those that reflect changes in the application semantics, which conventionally have called for schema redesign. Rather, these additions will largely call for the re-expression, in more convenient terms, of information already represented in the schema. This can be achieved by the disciplined use of redundancy in the conceptual and external schemas.

Much has been made in the data management literature of the evils of redundancy in databases; in fact, the success of database management systems has largely been fueled by their ability to limit data redundancy. However, we would make a plea for the virtues of <u>controlled redundancy</u>. The potential desirability of redundancy at the physical level, in

which alternate representations are maintained for a data item so that it may be accessed in different ways, has long been recognized. We submit that an analogous situation obtains at the logical level as well. Several users (or even an individual one) may associate multiple meanings with a single item of information and use it in different ways. Therefore, it is appropriate for this information to appear in several places in the schema, corresponding to the several ways in which it is viewed. Although this does introduce redundancy into the schema, it also provides the user with the information where he wants it. For example, it could be useful for information about a ship's captain to be available both in the record describing the ship as well as in the captain's personnel record. Even though each of these occurrences is redundant, and could be obtained from the other, the presence of both in the schema enables the user to more conveniently phrase certain queries involving ships and captains than he could if only one of them were available.

In this approach, the conceptual and external schemas may contain the same information in several places. In order to control and manage this redundancy, the database system will have to be aware of the logical relationships among seemingly distinct components of the schema. This knowledge should itself be explicitly contained in the schema, so that it be accessible to users.

In general, redundancy in the schema may involve more complex relationships among schema components than the simple equality of two fields in different records. It must provide for many kinds of situations in which some aspect of the database is partly determined by some other aspect. It should also be observed that logical redundancy does not necessarily have any implications at the physical design level. Logically redundant data might indeed be replicated at the physical level as well, or it might not; in the latter case, the DBMS will know to access a single physical data representation for references to any of its logical manifestations. Whether or not the data is physically replicated is an issue of

efficiency, turning on whether the costs of maintaining two versions of the same data item is counterbalanced by the access efficiency two copies provide.

The concept of controlled logical redundancy has two applications. It can be employed to enable the DBA to produce a "base" schema that is a more natural static model of the application and it can also be used to tune the schema to the changing characteristics of its usage pattern. In the first instance, logical redundancy supports a flexible and "relativist" view of the database. When there exist two equally valid ways of viewing a unit of information, it is inappropriate to force on the database designer the necessity of choosing between them; alternative representations of the information belong in the schema. In the second case, the needs of particular uses of the database can be accommodated by the definition of additional, "derived" components of the schema, redundant re-expressions of existing information that support the convenient programming of certain types of transactions. This part of the schema will evolve and grow as the uses of the database change. As users find themselves needing information that is not easily accessible in the current schema, the DBA will add new structures to meet these new needs, while leaving intact the existing ones.

This model of schema evolution by means of growth and derived information can simplify, decentralize, and facilitate the process by means of which a schema adapts to meet the changing characteristics of its operating environment. The addition of new structures to the schema, which are restatements of information already contained there, can be an easy to perform and relatively benign operation. As such, it need not be performed solely by a central master DBA, but could be done by local DBA's, who are principally concerned with one set of applications, or even by users themselves. Because these changes do not affect existing structures, there is little danger of disturbing the integrity of existing applications that depend on the old schema. This model coincides well with the notion of a federated

database introduced above. The utility and utilization of the database can also be enhanced as a result of this approach. In the contemporary view of the development of a schema, the database design evolves in a series of discrete stages. The DBA selects a schema, which remains operative until pressures mount for its change. These changes are then made and a revised schema defined, which reigns until the next redesign point. The flaw in this model is that it fails to recognize the fact that in this situation there are two adaptive systems at work: not only does the schema adapt to the users' needs, but the users adapt to what the schema provides. To a significant extent, users conceive their needs and desires in terms of the feasible; their thinking is influenced by the structure of the existing schema. This natural tendency is reinforced by the slow rate of change implied by periodic redesign points. If, however, the design process becomes a continuous and ongoing one, then no reigning design becomes so entrenched as to strongly influence users' needs and perceptions; the result will be a more effective information system. Changes in the application semantics of a database and in its usage pattern are nearly continuous, and so the response of the schema to them should be likewise. To freeze a design over a long period of time will inevitably force users to conform to a view of the database that is not convenient or suitable for them. By lowering the threshold of complexity and administrative overhead necessary to add to the schema, our approach can increase the frequency with which these changes are made and consequently improve the effectiveness of the information system as a whole.

To carry this approach even further, the notion of a fixed and well-defined schema to which data and its users must conform may become obsolete. In current systems, a data item must always be viewed as an instance of a construct in the schema; this binding is made at the time the data item is created (entered into the database), and is difficult (often impossible) to change. Greater flexibility in this regard will be needed to provide the kinds

of easy to use facilities needed in personal database environments. In such environments, it is desirable to be able to enter data into the database before a full schema of its description has been developed; even after a schema has been developed, it should be possible to insert data items that only partially conform to it; and the binding of a data item to its description should be made very flexible and amenable to change. In other words, the schema should conform to the data as much as the data conforms to the schema. Our model of database evolution can partially support this goal, by allowing for the dynamic definition of derived information in the schema, so that the description to be associated with a data item can be made to depend on its contents. Additional facilities will have to be developed to fully achieve this goal, and its dual in terms of retrieval: allowing the user to obtain data from the database without a full knowledge of the schema, or even of those parts of it that describe the data items he seeks. A limited amount of research has already been conducted in this area [Sagalowicz 1977]. New approaches to the structure of the interface between the user and the database system will be necessary to provide the needed level of functionality; some ideas in this direction are given in [McLeod 1978].

5. INFORMATION MODELLING

In the foregoing sections, we have discussed a number of new architectural directions for future database management systems. In order to meet these goals, a database system will have to make use of a high level specification of the information content and conceptual structure of the databases that it manages. Such a semantics-based schema is needed in order to support flexible and convenient user access to the database; to provide for the communications interface between components of a federated database; and to broaden the functional capabilities of a database management system. The character of the database description and structuring formalism (conceptual data model) that is used to

specify this schema will have a major impact on the capabilities and effectiveness of the DBMS. In this section, we explore the implications of our proposed DBMS architecture for the design of the conceptual data model in which the semantic schema will be expressed. A number of such modelling mechanisms have already been proposed [Abrial 1974, Bachman 1977, Chen 1976, Hammer+McLeod 1978, Kent 1978, Pirotte 1977, Roussopoulos 1977, Schmid+Swenson 1975, Senko 1975, Senko 1977, Wong+Mylopoulos 1977]. Our goal here is not to propose yet another one, but to develop some of the needed characteristics of such a model from the functional goals that we have outlined. We believe that there are a number of features that are essential for an effective conceptual data model, most of which are not provided by contemporary data models.

1. The modelling facilities of the data model should be solely concerned with capturing the meaning of the database, and should avoid all issues of database representation. The conceptual (or external) schema should be designed to optimize the understandability and usability of a database; the implementation and storage structures actually used to implement the database should be separately specified. This frees the users and applications from the need to understand the intimate details of a database's storage structures, and results in the well-known benefits of data independence and abstraction: increased understandability, maintainability,and evolvability.

Contemporary data models include many representation-oriented features. Essentially all data models in common use today are descendants of the "record" approach to database organization [Kent 1979]. As such, these data models represent compromises between the desire to ease the database management tasks of users and application programs, and the need to provide efficient database storage and manipulation facilities. It might be argued that the search for a truly representation-free data model is a never-ending one, in the sense

that any model is by definition only an approximation to reality; nonetheless, it is necessary to capture much more semantic information in a schema than is possible with contemporary data models. For example, the CODASYL DBTG data model [Codasyl 1971, Manola 1978] includes many features motivated by and concerned with achieving an efficient implementation of the database. Although the information expressed by these features is obviously important to database system performance and must be captured somewhere, it does not belong in the conceptual schema. Even though the relational data model does avoid issues of purely physical data storage [Codd 1970, Chamberlin 1976], relational structures are still record-oriented: relational DBMSs force an application environment to be modelled in terms of simple nonhierarchic logical records, each of which consists of a collection of simple values. Furthermore, in most relational implementations, the physical structure of a database is analogous to, and largely limited by, its relational schema. Even "logical" records are too representational for use in a conceptual schema: higher level semantics-based structures are required.

2. It is essential to minimize the gap between the nature of a database conceptual schema and the users' conceptions of the aspects of the application environment that are modelled in the database. Although a conceptual data model is artificial by its very nature (since it deals with the descriptions of things and not things themselves), the constructs of the conceptual data model should enable the database schema to be specified with structures that users can understand. The translation that a user must make from his own model of the application environment to the model specified in the database schema should be as direct as possible. Unfortunately, conventional data models do not adequately satisfy this criterion; their record orientation prohibits them from naturally modelling an application environment.

Specifically, we believe that a conceptual data model must facilitate the modelling of things (entities), associations among things (relationships), collections of things (classes), and structural interconnections among the collections (interclass connections). Unlike conventional data models, where record-oriented structures are used to accommodate all of these concepts, the conceptual data model should directly and distinctly capture each of these types of information. For example, in the relational data model, a single mechanism (the relation) is used to model a collection of entities, to express an association among entities, and so forth. This semantic overloading of the relation makes it difficult for a user to determine the meaning and purpose of a relation, and obscures the meaning of a database as a whole. Assuring that a relational schema is in an appropriate normal form does guarantee its freedom from various "updating anomalies", but does not necessarily contribute to its understandability and utility; the "formal semantics" of functional dependencies employed by the normalization process are unrelated to the natural semantics that ought to be captured by, and manifest in, the schema. Ad hoc solutions, such as the judicious choice of relation and column names or copious documentation, are not adequate, since they can only supplement the essential structure of the schema. While the Codasyl DBTG data model does provide different features to model different semantic constructs (records to model entities and "sets" to model associations), the model also exhibits degrees of modelling freedom that do not involve the meaning of data, and so complicate the design process and obscure the resultant schema. For instance, a multi-valued attribute of an entity can be modelled either as a repeating group or by using a DBTG "set"; the choice between these modelling methods is essentially arbitrary, in the sense that it does not involve the meaning of the information. The features of the conceptual data model ought to be so closely based on application semantics that such choices of representation need not be made.

Another important shortcoming of conventional data models is that they do not

provide unified approaches to important semantic constructs. For instance, we note that neither the relational, the DBTG, nor the hierarchical data model provides a natural and integrated approach to the modelling of relationships. For example, in each of these models, one-to-many and many-to-many associations are treated quite differently. A one-to-many association is typically modelled by a DBTG "set", by a hierarchical parent-child link, or by a relational attribute of the latter entity (the "many" side of the one-to-many association, since this results in a nonrepeating attribute value [Kent 1978]). By contrast, many-to-many associations must be modelled in a different way, by introducing DBTG linking records, multiple hierarchical parent-child links, or an association relation. The essential problem here is that nonuniform and ad hoc mechanisms are used to accommodate a single simple semantic concept, viz., an association among entities.

An important goal of a conceptual data model is to encourage and even permit only meaningful interpretations and manipulations of a database. Conventional data models do not adequately achieve this. The relational data model allows database relations to be "joined", regardless of whether or not such joins make semantic sense. The issue is that all specification of information combination is provided by the user or application at retrieval time, rather than being explicitly present in the schema. The Codasyl DBTG data model provides some capability to specify which links among records may be followed (via "set" types), but this specification is not strict and is also tied up in a complex framework of implementation-oriented considerations.

3. A database conceptual schema must support a "relativist" view of the meaning of a database; a conceptual data model must allow a schema to reflect alternative ways of looking at information. There is in general no fundamental and basic way to divide the information in a database into individual "facts". On the contrary, there are typically many

points of view that are equally valid and significant. Moreover, even an individual user or application does not necessarily have a single way of looking at each fact. These multiple perspectives multiply over time as the usage of the database evolves. In order to accommodate multiple ways of looking at the same information and to support the evolution of new ways of viewing the same basic information, a conceptual schema must be flexible and logically redundant.

Conventional data models do not accommodate this inherent need for relativism and flexibility. They demand that a single structural organization be imposed on the data, one that inevitably carries along with it a particular interpretation of the data's meaning. This meaning may not be acceptable to all users of the database and may become entirely obsolete over time. For example, an association between two entities can legitimately be viewed as an attribute of the first entity, as an attribute of the second entity, and as an entity itself; thus, an assignment of an officer as the captain of a ship can be viewed as an attribute of the ship (current captain), as an attribute of the officer (current ship), and as an entity (assignment). The schema must make all three of these interpretations equally natural and direct. Therefore, the conceptual data model must provide a specification mechanism that simultaneously accommodates all three of these ways of looking at an association.

Conventional data models do not adequately accommodate relativism. Fixed representations must be chosen for the application constructs that are being modelled. View and subschema mechanisms do not materially affect this situation: first, because of their limited capability for restructuring the base schema; and second, because within a given view, flexibility is once again lost.

A consequence of the observation of the primacy of the principle of relativism is that there is in general no clear distinction between the concepts of entity, association, and attribute. Data models that require the designer to sharply distinguish among these

concepts (such as [Chen 1976, Pirotte 1977]) thus impose an undesirable degree of rigidity on the conceptual schema.

4. A conceptual data model should allow entities to logically represent themselves in a database, rather than requiring users to explicitly use some surrogate identifiers to establish inter-entity logical connections. For example, when viewing a relationship between entities as an attribute of one of the involved entities, the value of the attribute should be the other entity, rather than some identifier for it.

One of the most important reasons for allowing entities to stand for themselves is that this makes it possible to directly reference an entity from a related one. In conventional data models, this is not possible; it is necessary to cross reference between related entities via identifiers. For example, it may be necessary to obtain from a ship entity the entity that models the ship's current captain. Thus, we should like to be able to define an attribute "Captain" that applies to any ship, and whose value is an officer. To model this information using the relational data model, it is necessary to select some identifier for officers (e.g., their last name or their identification number) to stand as the value of the "Captain" attribute of ships. Specifically, we might have a relation SHIPS, one of whose attributes is Officer-name, and a relation OFFICERS, which has Officer-name as a candidate key. Then, in order to find the information on the captain of a given ship, it is necessary to explicitly join relations SHIPS and OFFICERS on Officer-name; that is, an explicit cross reference via identifiers is required. This forces the user to deal with an extra level of indirection and to consciously apply joins. (Joins are known to be semantically awkward constructs and are difficult for users to apply [Reisner 1977].) The "set" mechanism of the Codasyl DBTG data model does to a limited extent support the direct interconnection of entities (if entities are modelled by records); however, as noted above, this mechanism

lacks flexibility.

The conceptual data model must support the distinction between an entity and its identifier. A failure to recognize this distinction results in serious problems when changes are made to the entity. Changing a name of an entity does not usually imply that a new entity has been created, but rather that an existing entity has undergone a change; other aspects of the entity may remain unchanged. It is well known to programming language designers that failing to adequately accommodate the distinction between a variable and its value has a serious adverse impact on programming language semantics and consequently on program understandability; an analogous observation can be made for database entities and their identifiers.

In consequence of the fact that conventional data models require surrogates to be used to stand for entities in connections among entities, important types of semantic integrity constraints on a database are not directly captured in a conceptual schema. If these semantic constraints are to be expressed and enforced, additional mechanisms must be provided to supplement conventional data models [Eswaran+Chamberlin 1975, Hammer+McLeod 1975, Hammer+McLeod 1976, Stonebraker 1974]. The problem with this approach is that these supplemental constraints are at best ad hoc, and do not integrate all available information into a simple structure. For example, it is desirable to require that only captains who are known in the database be assigned as officers of ships. To accomplish this in the relational data model, it is necessary to impose the supplemental constraint that each value of attribute Captain-name of SHIPS must be present in the Captain-name column of relation OFFICERS. If it were possible to simply state that each ship has a captain attribute whose values is an officer, this supplemental constraint would not be necessary.

5. A conceptual data model must support the distinction between what is permissible and what is currently the case in the database. For example, for a class (C) of entities each of which has some attribute (A), it is important to distinguish the possible values of A from the set of current values of A.

Although limited in power, the relational data model does have a mechanism for accommodating the distinction of possible versus current values in a database. In particular, domains are intended to model possible sets of values, while relation columns contain current values [McLeod 1977]. Unfortunately, domains are collections of atomic (e.g., string) values, and it is not possible to state that the possible values of an column are to be chosen from some nonatomic set of values (that model entities). For example, calendar dates can be specified as a domain (a collection of strings), or can be modelled by a relation involving days, months, and years. If modelled as a domain, dates can then be specified as the possible values of some column. However, if specified as a relation, it is not directly possible to require the values of some column to be selected from the tuples of the relation modelling dates. Thus, the distinction of domains and relations is inadequate for differentiating permissible values from current values.

6. The conceptual data model must provide constructs to capture the various types of possible logical structural interconnections among collections of entities. The subset connection is particularly important, since in general, an entity may belong to many classes. Conventional data models do not allow this: a record is a member of only one file; a tuple belongs to a single relation; a DBTG record is an instance of one record type. In consequence, it is difficult in these models to subdivide collections of entities into meaningful units. Moreover, since attributes are defined over such collections (e.g., each record in a file has the same fields), it is inconvenient to accommodate attributes that apply only to some

subset of the entities in a collection. For example, all oil tankers are ships; oil tankers have all the attributes of ships, and may further have attributes that other ships in general do not. (In contemporary systems, such situations are often handled by associating the tanker-specific attributes with all ships and entering null or blank values for them in ships that are not tankers. Needless to say, such ad hoc mechanisms are cumbersome and prone to error.)

This is not meant to imply that a simple hierarchy of classes of entities is sufficient. For example, it may be necessary to model the class of all ships, the class of all oil tankers, the class of all Liberian ships, and the class of all Liberian oil tankers. Thus, the conceptual data model must allow collections of entities to be simultaneously related to a number of other collections in the database; this is yet another instance of the need for supporting several relative points of view.

Second-order collections of entities must also be accommodated in a conceptual schema. For example, it is necessary to be able to model entities that are themselves collections of other entities, e.g., a group of ships, such as a convoy. Moreover, a mechanism is required for modelling an entity that is a generalization [Hammer+McLeod 1978, Lee+Gerritsen 1978, Palmer 1978, Smith+Smith 1977a, Smith+Smith 1977b] of another kind of entity. For example, the class of all ship types can be defined as a collection of generalizations of ships; the members of the class of all ship types have their instances in the class of all ships [McLeod 1978]. Conventional data models provide no direct capability for modelling these various types of second order collections.

7. The conceptual data model should embody a unified approach to the database schema and the data itself. Using conventional data models, the structure of a database is specified in the schema, and the data is expressed as instances of that structure. The data and the schema are disjoint, except for the fact that the schema describes the data instances. Some

information is thus expressed in the schema, while other is recorded as data. Different mechanisms are used to extract data from the database and to manipulate the schema. Queries that can be answered by strictly examining the data are normally supported by the DBMS, while those that also require examination of the schema must be performed in part by the user. Since conventional schemas do not capture the semantics of the application environment, this means that the user must consciously exploit his external knowledge to answer some questions.

Some database systems are supplemented by data dictionary facilities [Grandwell 1975, Uhrowczik 1973]. Unfortunately, these facilities do not provide many capabilities for capturing the meaning of a database. Data dictionaries are typically used to record "factorable" facts, those that are true of an entire collection of data items [Kent 1978]. However, the dictionaries are not integrated with the data manipulation facilities of the DBMS; they are mild extensions of the database conceptual schema.

We believe that it is essential to begin to remove the barrier between schema and data, and between schema and data dictionary. The conceptual schema should provide a unified and integrated mechanism that permits database concepts to be expressed and manipulated regardless of whether they are "structural" or "factual"; again, whether a given item of information is expressed as structure or fact is a question of perspective. For example, a collection of names of home ports may be defined as a class of entities to which new names might be explicitly added, and from which existing names might be explicitly removed. Alternatively, the collection of home port names could be defined as some given subset of all possible entity names. In the former approach, the port names are captured as data, while in the latter strategy, the schema is used to express the information. The conceptual schema must unify these alternatives, by recognizing that the same information can be viewed in two ways.

8. Derived information must be directly accommodated in the conceptual data model. As noted above, it is essential for the structure of a database to capture the alternative ways of looking at the same facts. By incorporating derived information into a database schema, the database's structure becomes more flexible, and its manipulation is simplified. Moreover, the inclusion of frequently used items of derived information in a database schema makes that information more directly accessible, and thus simplifies data manipulations that involve it. The expression and control of logical redundancy (in the form of derived information) must therefore be directly supported by the conceptual data model.

We believe that the features of a conceptual data model presented above will be essential in a future generation of database management systems, in order to meet the growing demands being placed on database management technology. The conceptual data model is the focal point of the critical trends described above: the unification of data, knowledge, and application domain information; the extension of the functional capabilities of a DBMS, so that the DBMS holds a position of primacy within the data-intensive application system; the recognition of the importance of the life cycle of database design, redesign, and evolution; the establishment of a basis for accommodating communication and sharing among both strongly and loosely coupled applications; and the development of unified, advanced DBMS interface facilities. The development of conceptual data models with the aforementioned features is an important research goal. We have developed an initial data model with these objectives in mind [Hammer+McLeod 1978, McLeod 1978], but it represents only a first step.

6. CONCLUSIONS

We have reconsidered a number of fundamental assumptions that underlie the contemporary view of databases and database management. In particular, we have argued that the notion of an integrated database may no longer be an appropriate one and that a database should be viewed as a loose federation of semi-independent components; that a database management system should not be viewed as a subsidiary component of an information system but as its centerpiece and that in consequence the functions and capabilities of a DBMS ought to be extended to incorporate many activities currently outside its domain; and that controlled logical redundancy in a database can be used to enhance the database's usability and to manage its life cycle. A theme underlying all of these issues is that of using a conceptual data model to specify a semantic schema for a database that will capture more of the meaning of the application than is currently represented by conventional data models.

Our analysis represents just the beginning of a potentially new direction in database management. We have determined only some of the architectural desiderata for future database systems; other departures from current organizations may also need to be taken, and much work needs to be done to reduce the principles that we have adumbrated to a concrete architectural specification. We have designed a first version of a conceptual data model that meets some of the criteria outlined above, but it too is neither complete nor entirely adequate. In future efforts, we hope to further explore the issues that have been raised here.

REFERENCES

[Abrial 1974]
Abrial, J. R., "Data Semantics", in J. Klimbie and K. Koffeman (eds.), **Data Base Management**, North Holland, 1974.

[Bachman 1977]
Bachman, C. W., "The Role Concept in Database Models", *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.

[Buneman+Morgan 1977]
Buneman, O. P. and H. L. Morgan, "Implementing Alerting Techniques in Database Systems", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.

[Chamberlin 1976]
Chamberlin, D. D., "Relational Data-Base Management Systems", *Computing Surveys*, Volume 8, Number 1, March 1976.

[Chen 1976]
Chen, P. P. S., "The Entity - Relationship Model: Toward a Unified View of Data", *ACM Transactions on Data Base Systems*, Volume 1, Number 1, Pages 9-36, March 1976.

[Codasyl 1971]
Codasyl Committee on Data System Languages, *Codasyl Data Base Task Group Report*, ACM, New York NY, 1971.

[Codd 1970]
Codd, E. F., "A Relational Model for Large Shared Data Banks", *Communications of the ACM*, Volume 13, Number 6, June 1970.

[Eswaran+Chamberlin 1975]
Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Database Integrity", *Proceedings of the First International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.

[Grandwell 1975]
Grandwell, J. L., "Why Data Dictionaries?", *Database*, Volume 6, Number 2, Pages 15-18, 1975.

[Hammer+McLeod 1975]
Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System", *Proceedings of the First International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.

[Hammer 1976]
Hammer, M. M., "Data Abstractions for Data Bases", *Proceedings of ACM SIGMOD/SIGPLAN Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.

[Hammer+McLeod 1976]
Hammer, M. M. and D. J. McLeod, "A Framework for Data Base Semantic Integrity", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.

[Hammer+McLeod 1978]
Hammer, M. and D. McLeod, "The Semantic Data Model: A Modelling Mechanism for Data Base Applications", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.

[Hammer 1979]
Hammer, M., "Research Directions in Data Base Management", in P. Wegner (editor), **Research Directions in Software Technology**, MIT Press, 1979.

[Kent 1978]
Kent, W., **Data and Reality**, North Holland, 1978.

[Kent 1979]
Kent, W., "Limitations of Record-Based Information Models", *ACM Transactions on Database Systems*, Volume 4, Number 1, Pages 107-131, March 1979.

[Lee+Gerritsen 1978]
Lee, R. M. and R. Gerritsen, "Extended Semantics for Generalization Hierarchies", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.

[Liskov+Snyder+Atkinson+Schaffert 1977]
Liskov, B., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU", *Communications of the ACM*, Volume 20, Number 8, Pages 564-576, August 1977.

[Manola 1978]
Manola, F., "A Review of the 1978 CODASYL Database Specifications", *Proceedings of the Fourth International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.

[McLeod 1977]
McLeod, D. J., "High Level Definition of Abstract Domains in a Relational Data Base System", *Journal of Computer Languages*, Volume 2, Number 3, 1977.

[McLeod 1978]
McLeod, D., *A Semantic Database Model and Its Associated Structured User Interface*, Technical Report, MIT Laboratory for Computer Science, Cambridge MA, 1978.

[Nijssen 1976]
Nijssen, G.M., "A Gross Architecture for the Next Generation of Database Management Systems", in G.M. Nijssen (editor), **Modelling in Data Base Management Systems**, North-Holland, 1976.

[Palmer 1978]
Palmer, I., "Record Subtype Facilities in Database Systems", *Proceedings of the Fourth*

International Conference on Very Large Data Bases, West Berlin, West Germany, 13-15 September 1978.

[Parnas 1972]
Parnas, D., "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, Volume 15, Number 12, Pages 1053-1058, December 1972.

[Pirotte 1977]
Pirotte, A., *The Entity - Property - Association Model: An Information-Oriented Data Base Model*, Technical Report, M.B.L.E. Research Laboratory, Brussels, Belgium, 1977.

[Reisner 1977]
Reisner, P., "The Use of Psychological Experimentation as an Aid to Development of a Query Language", *IEEE Transactions on Software Engineering*, Volume SE-3, Number 3, May 1977.

[Roussopoulos 1977]
Roussopoulos, N., "ADD: Algebraic Data Definition", *Proceedings of Sixth Texas Conference on Computing Systems*, Austin TX, 14-15 November 1977.

[Sagalowicz 1977]
Sagalowicz, D., "IDA: An Intelligent Data Access Program", *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.

[Schmid+Swenson 1975]
Schmid, H. A. and J. R. Swenson, "On the Semantics of the Relational Data Model", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.

[Schmidt 1977]
Schmidt, J. W., "Some High Level Language Constructs for Data of Type Relation", *ACM Transactions on Database Systems*, Volume 2, Number 3, Pages 247-261, September 1977.

[Senko 1975]
Senko, M. E., "Information Systems: Records, Relations, Sets, Entities, and Things", *Information Systems*, Volume 1, Number 1, Pages 3-14, 1975.

[Senko 1977]
Senko, M. E., "Conceptual Schemas, Abstract Data Structures, Enterprise Descriptions", *Proceedings of ACM International Computing Symposium*, Belgium, April 1977.

[Smith+Smith 1977a]
Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation", *Communications of the ACM*, Volume 20, Number 6, Pages 405-413, June 1977.

[Smith+Smith 1977b]
Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, Volume 2, Number 2, Pages 105-133, June 1977.

[Stonebraker 1974]

Stonebraker, M. R., *High Level Integrity Assurance in Relational Data Base Management Systems*, Electronics Research Laboratory Report ERL-M473, University of California, Berkeley CA, 16 August 1974.

[Stonebraker 1977]
Stonebraker, M., "Database Languages", *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.

[Uhrowczik 1973]
Uhrowczik, P. P., "Data Dictionary/Directories", *IBM Systems Journal*, Number 4, 1973.

[Wong+Mylopoulos 1977]
Wong, H. K. T. and J. Mylopoulos, "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management", *Infor*, Volume 15, Number 3, Pages 344-382, October 1977.